**Styled Components**

1. **Introduction**
2. **What and why is React styled component?**

React Styled Components is a popular library for styling React applications using a technique called "CSS-in-JS" (CSS in JavaScript). It allows developers to write CSS code as JavaScript template literals, which are then used to style React components. This approach offers several advantages:

1. Scoped Styles: Styled Components generate unique class names for each component, ensuring that styles do not leak or clash with other components. This helps in maintaining a modular and encapsulated codebase.
2. Dynamic Styles: You can easily create dynamic styles by interpolating JavaScript variables or props directly into your styled components, making it flexible and responsive.
3. Theming: Styled Components makes it straightforward to implement theming in your application by defining themes as JavaScript objects and then accessing theme values within your styled components.
4. CSS Preprocessing: Styled Components supports the use of CSS preprocessors like Sass or Less, allowing you to write more structured and maintainable styles.
5. Server-Side Rendering (SSR): It integrates well with server-side rendering solutions, ensuring that styles are correctly rendered on the server and hydrated on the client, which improves performance and SEO.

React Styled Components has gained popularity for its ease of use, readability, and the ability to solve many styling-related challenges in React applications. However, there are other CSS-in-JS libraries available as well, each with its own set of features and benefits.

1. **Benefits of react styled components.**
2. **Encapsulation**: Styled Components encourage a more modular and component-based approach to styling, improving code organization and maintainability.
3. **Readability**: CSS code is written as JavaScript template literals, which can be more readable and maintainable than traditional CSS.
4. **Flexibility**: You have full access to JavaScript's expressive power to create and compute styles, making it easy to handle dynamic styles and complex logic.
5. **Community and Ecosystem**: React Styled Components has a large and active community, offering many resources and third-party packages to extend its functionality.
6. **Server-Side Rendering**: SSR support ensures that styles are correctly handled during server-side rendering, improving performance and SEO.
7. **Developer Experience**: Many developers find the development experience with Styled Components to be pleasant and efficient.

Overall, React Styled Components is a powerful and flexible tool for styling React applications, offering a range of features that enhance code maintainability and make it easier to create dynamic and scoped styles.

* 1. **Installation**

**Une image contenant texte, capture d’écran, Police

Description générée automatiquement**

1. **Basics**
2. **Getting started**

styled-components utilises tagged template literals to style your components.

It removes the mapping between components and styles. This means that when you're defining your styles, you're actually creating a normal React component, that has your styles attached to it.

**Button.styles.js**

import styled from "styled-components";

export const StyledButton = styled.button`

  border: 2px solid #4caf50;

  background-color: "#4caf50";

  color: "#FFF";

  padding: 15px 32px;

  text-align: center;

  text-decoration: none;

  display: inline-block;

  font-size: 16px;

  cursor: pointer;

  transition: 0.5s all ease-out;

`;

**Button.js**

import { StyledButton } from "./Button.styles";

export default StyledButton;

**App.tsx**

<div className="App">

      <Button>Test </Button>

    </div>

1. **Adapting Props**

You can pass a function ("interpolations") to a styled component's template literal to adapt it based on its props.

**Button.styles.js**

import styled from "styled-components";

export const StyledButton = styled.button`

  border: 2px solid #4caf50;

  background-color: ${(props) =>

    props.variant === "outline" ? "#FFF" : "#4caf50"};

  color: ${(props) => (props.variant === "outline" ? "#4caf50" : "#FFF")};

  padding: 15px 32px;

  text-align: center;

  text-decoration: none;

  display: inline-block;

  font-size: 16px;

  cursor: pointer;

  transition: 0.5s all ease-out;

`;

**App.js**

 <Button variant="outline">Styled B

utton</Button>

1. **Extending styles**

Quite frequently you might want to use a component, but change it slightly for a single case. Now, you could pass in an interpolated function and change them based on some props, but that's quite a lot of effort for overriding the styles once.

To easily make a new component that inherits the styling of another, just wrap it in the styled() constructor. Here we use the button from the last section and create a special one, extending it with some color-related styling:

**Button.styles.js**

import styled from "styled-components";

export const StyledButton = styled.button`

  border: 2px solid #4caf50;

  background-color: ${(props) =>

    props.variant === "outline" ? "#FFF" : "#4caf50"};

  color: ${(props) => (props.variant === "outline" ? "#4caf50" : "#FFF")};

  padding: 15px 32px;

  text-align: center;

  text-decoration: none;

  display: inline-block;

  font-size: 16px;

  cursor: pointer;

  transition: 0.5s all ease-out;

`;

export const FancyButton = styled(StyledButton)`

  background-image: linear-gradient(to right, #f6d365 0%, #fda085 100%);

  border: none;

`;

**Button.js**

import { StyledButton, FancyButton } from "./Button.styles";

export default StyledButton;

export { FancyButton };

**App.js**

 <div className="App">

      <Button>Test </Button>

      <div>

        <br />

      </div>

      <Button variant="outline">Styled Button</Button>

      <div>

        <br />

      </div>

      <FancyButton as="a">Fancy Button</FancyButton>

    </div>

**As=”a” we will have link a**